Linux And Shell Scripting

Growth School

Agenda

1. What is Linux?
2. What is Shell Scripting?
3. Advantage of Shell Scripting
4. Disadvantage
5. Important Linux Commands
6. Shell Scripting Programming
7. Scheduling

What is Linux?

- **Operating System:** Linux is a family of open-source operating systems based on the Linux kernel. It provides the core functionality for interacting with hardware, managing resources, and running applications. Think of it as the foundation of your computer that allows other programs to function.
- **Diverse Functionality:** Linux encompasses a wide range of distributions (e.g., Ubuntu, Debian, CentOS) catering to different needs, from servers to desktops. It empowers users with diverse functionalities like file management, networking, security, and more.
- **Graphical and Command-Line Interfaces:** Linux offers both graphical user interfaces (GUIs) like desktops and command-line interfaces (CLIs) like shells.

What is Shell Scripting?

- **Programming Technique:** Shell scripting is a way to automate tasks and processes on Linux systems by writing scripts of commands that the shell interprets and executes.
- **Language and Interpreter:** Scripts are written in a specific language (e.g., Bash, Python) and interpreted by a shell program (e.g., Bash, Zsh) that understands and executes the commands.
- **Automation and Customization:** Shell scripting allows you to automate repetitive tasks, manage complex workflows, and customize system behavior without relying solely on manual commands.

Different Types of Shell in Linux

- **BASH (Bourne Again Shell):** Most Widely Used, Default Login shell in Linux and macOS, can be installed in Windows
- **CSH (C Shell):** C Shell language similar to C Programming Language
- **KSH (Korn Shell):** Based for POSIX, Each Shell does the same job but understand different commands and provide different built in functions

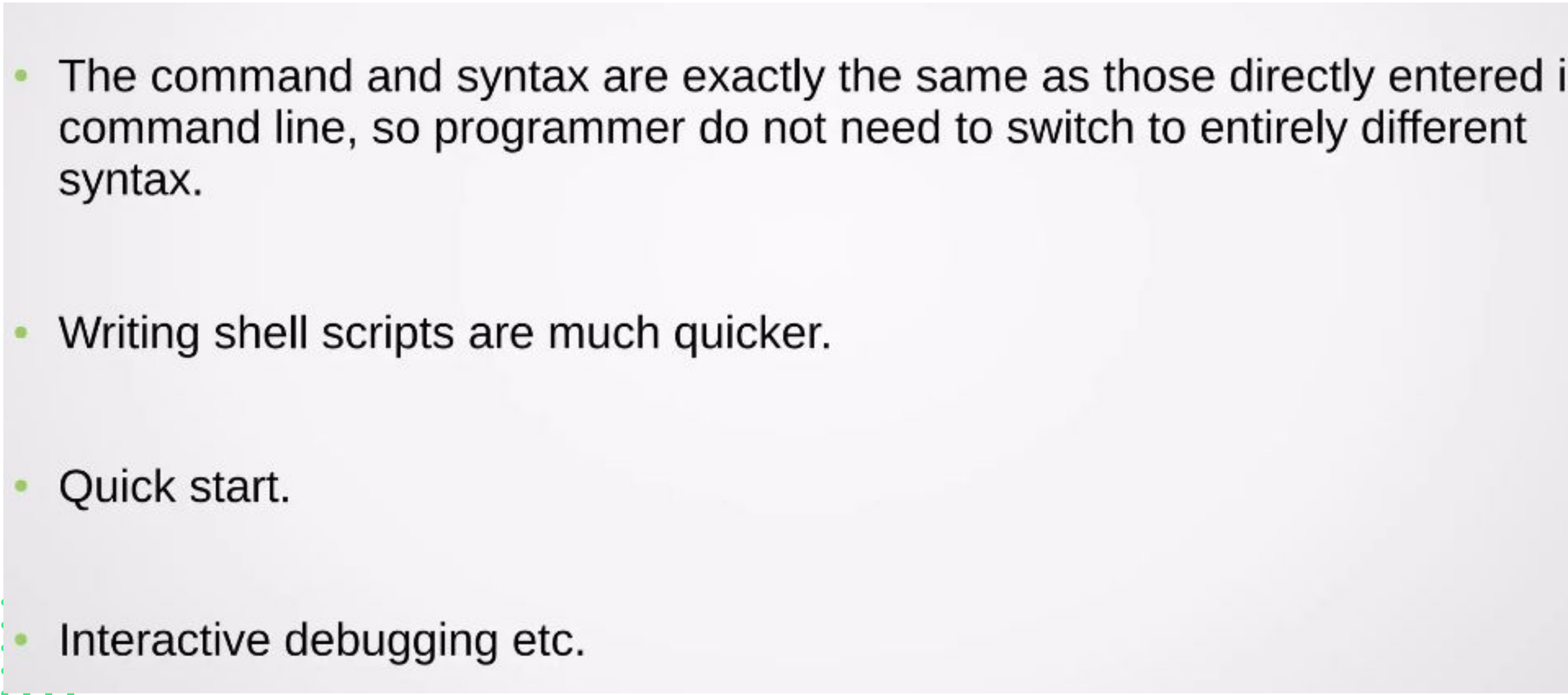
Key Differences:

- Scope: Linux is a complete operating system, while shell scripting is a technique within that system for automating tasks.
- Complexity: Managing an entire operating system like Linux requires deeper understanding, compared to writing script commands.
- Applications: Shell scripting offers practical benefits for data analysis, system administration, and automating processes specific to Linux environments.

Relationships:

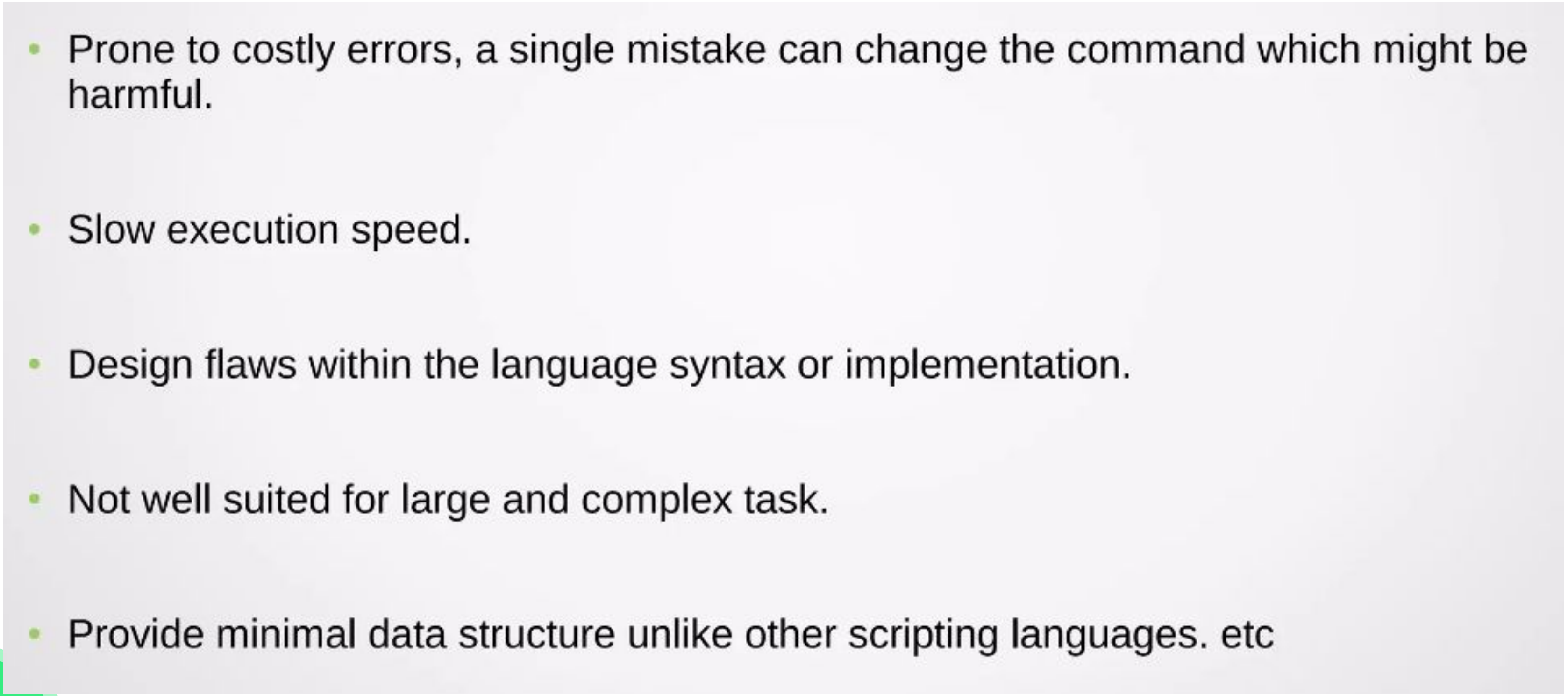
- Think of Linux as the stage and shell scripting as the play performed on that stage.
- Linux provides the environment, resources, and tools, while shell scripting leverages them to automate specific tasks or processes.

Advantage of Shell Scripting



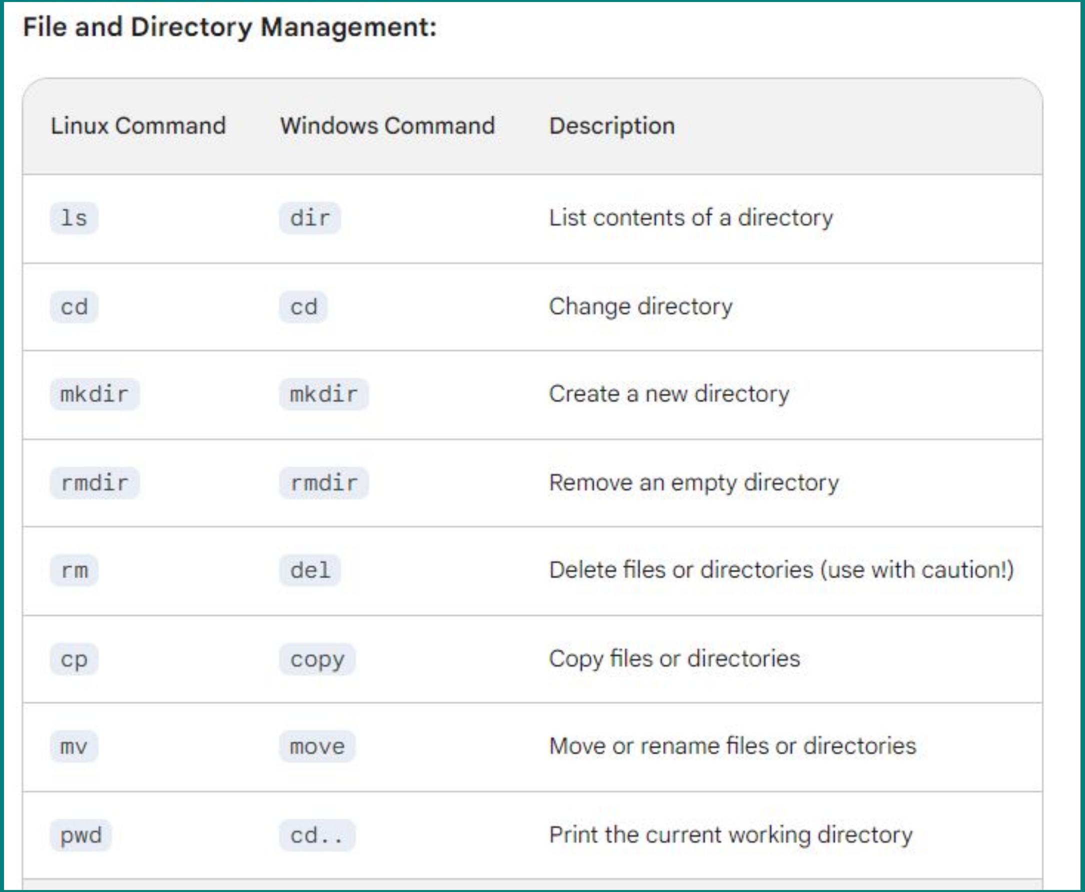
- The command and syntax are exactly the same as those directly entered i command line, so programmer do not need to switch to entirely different syntax.
- Writing shell scripts are much quicker.
- Quick start.
- Interactive debugging etc.

Disadvantage



- Prone to costly errors, a single mistake can change the command which might be harmful.
- Slow execution speed.
- Design flaws within the language syntax or implementation.
- Not well suited for large and complex task.
- Provide minimal data structure unlike other scripting languages. etc

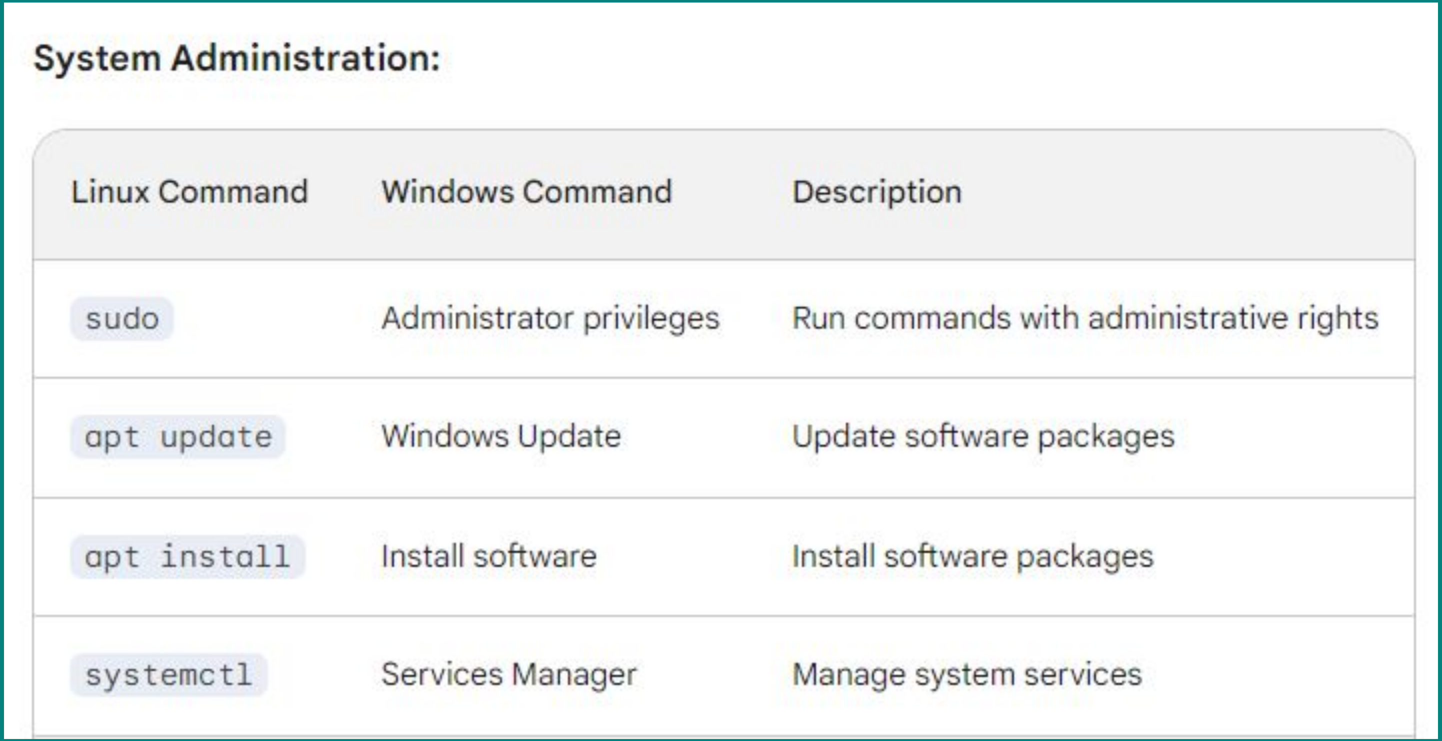
Important Linux Commands - Basic 1



File and Directory Management:

Linux Command	Windows Command	Description
`ls`	`dir`	List contents of a directory
`cd`	`cd`	Change directory
`mkdir`	`mkdir`	Create a new directory
`rmdir`	`rmdir`	Remove an empty directory
`rm`	`del`	Delete files or directories (use with caution!)
`cp`	`copy`	Copy files or directories
`mv`	`move`	Move or rename files or directories
`pwd`	`cd..`	Print the current working directory

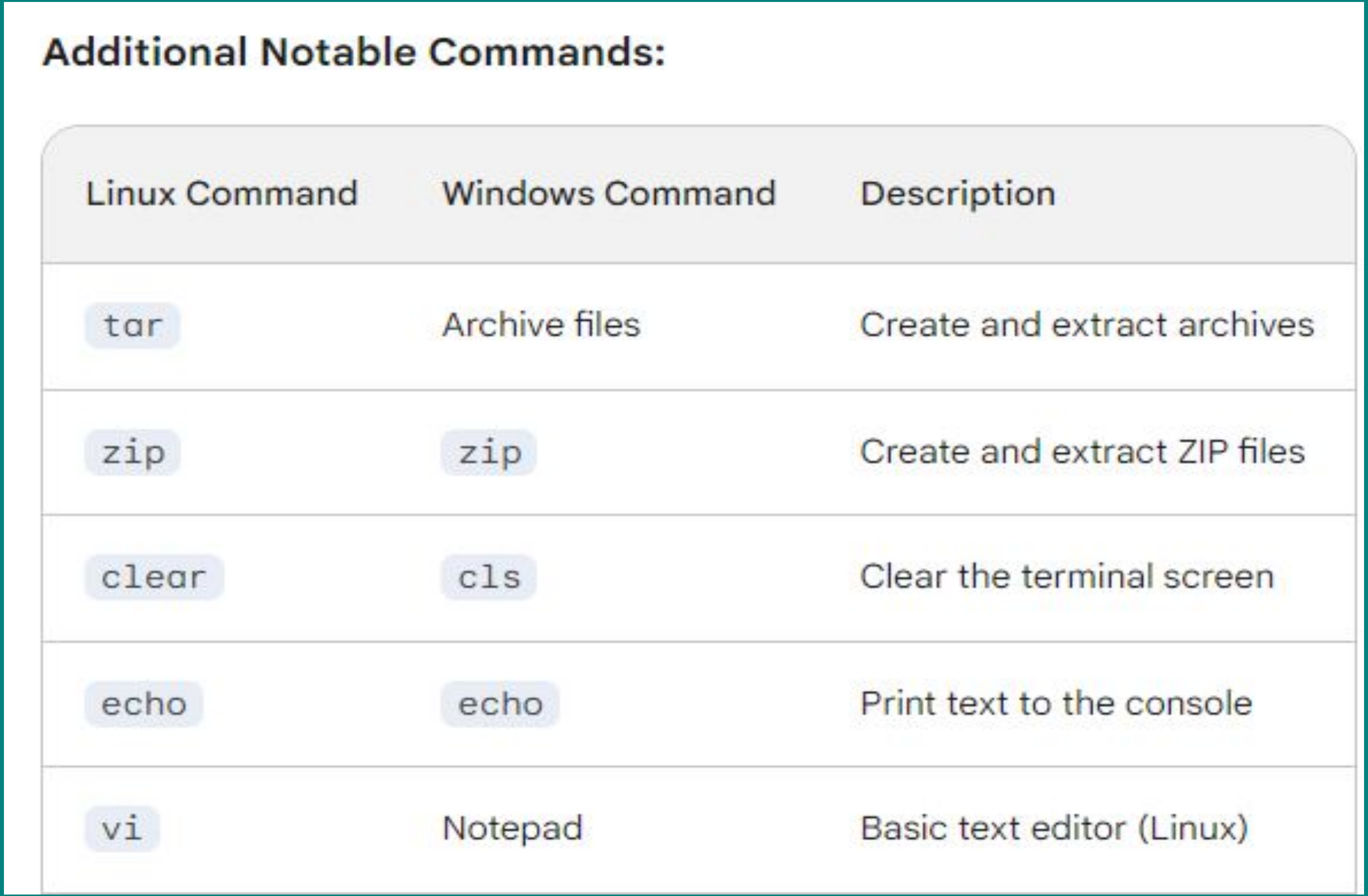
Important Linux Commands - Basic 2



System Administration:

Linux Command	Windows Command	Description
`sudo`	Administrator privileges	Run commands with administrative rights
`apt update`	Windows Update	Update software packages
`apt install`	Install software	Install software packages
`systemctl`	Services Manager	Manage system services

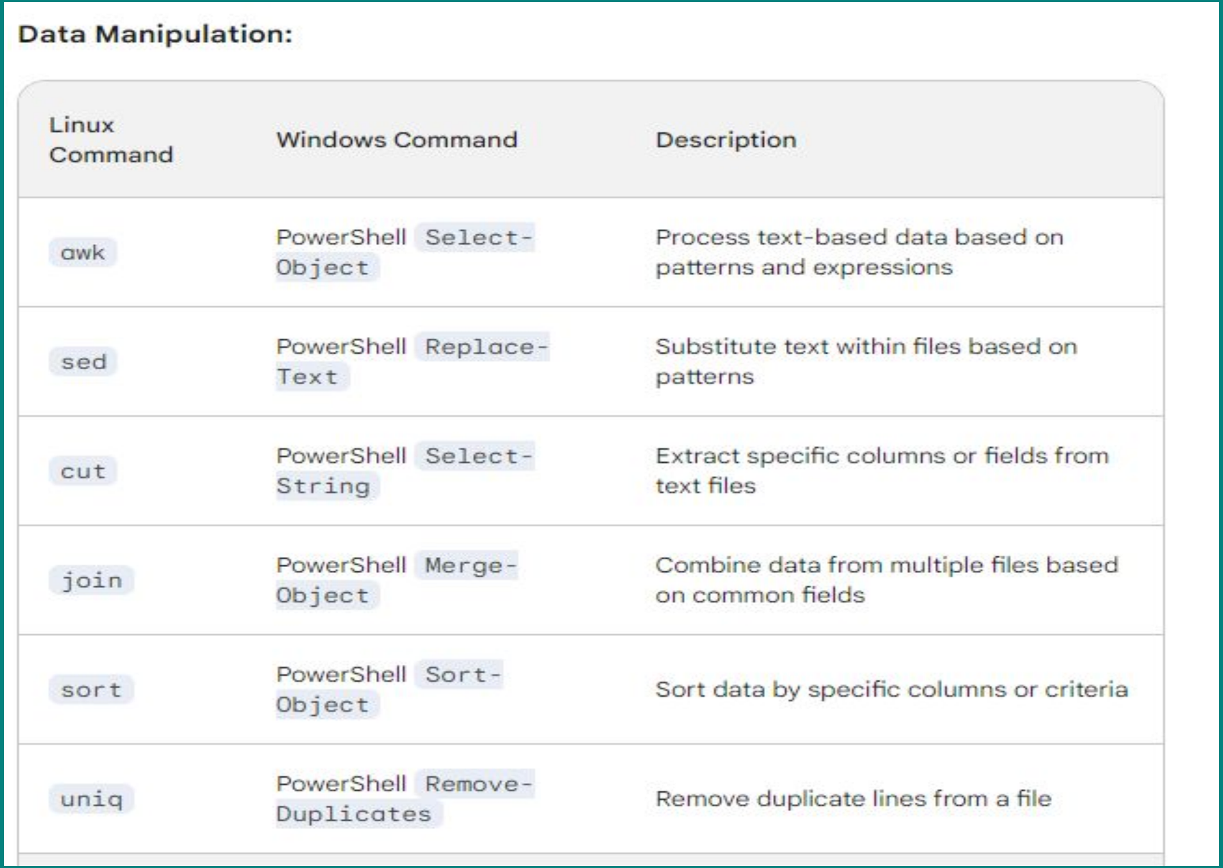
Important Linux Commands - Basic 3



Additional Notable Commands:

Linux Command	Windows Command	Description
`tar`	Archive files	Create and extract archives
`zip`	`zip`	Create and extract ZIP files
`clear`	`cls`	Clear the terminal screen
`echo`	`echo`	Print text to the console
`vi`	Notepad	Basic text editor (Linux)

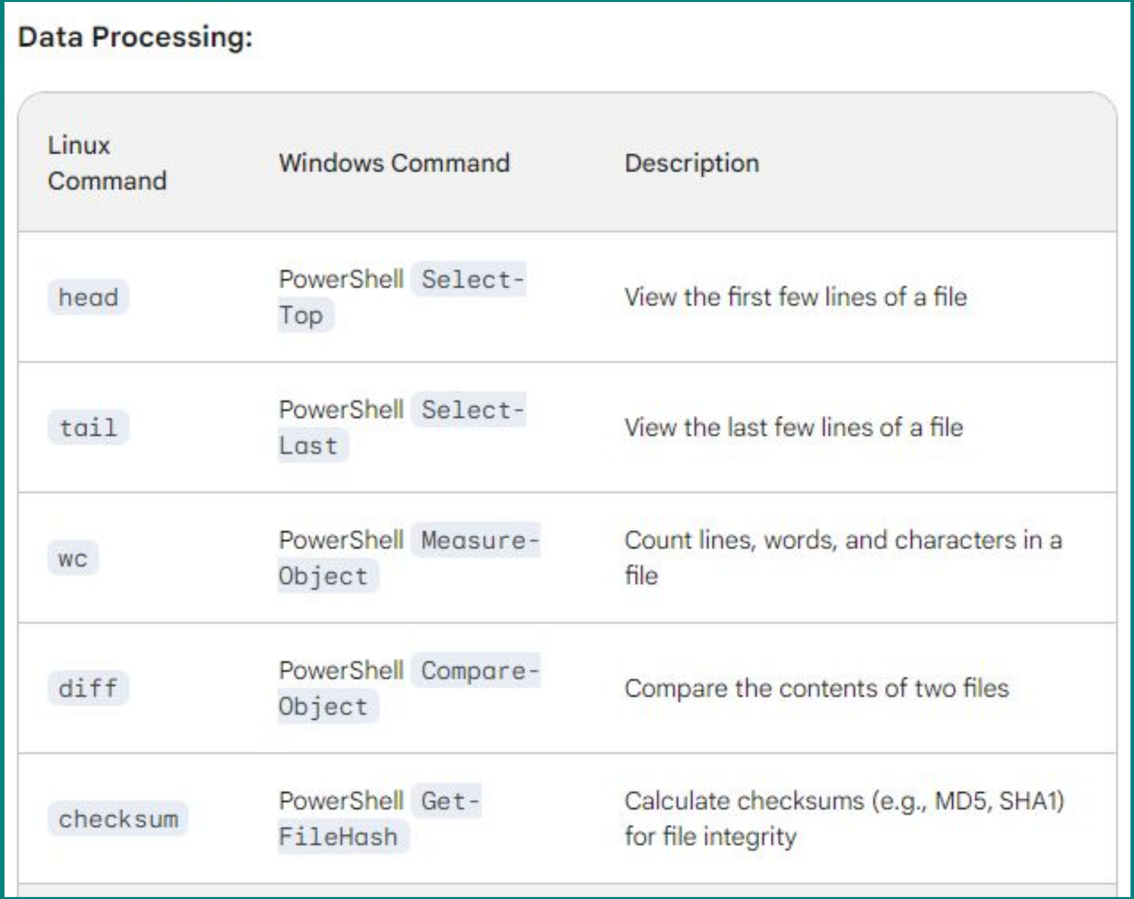
Important Linux Commands - Advance Part 1



Data Manipulation:

Linux Command	Windows Command	Description
`awk`	PowerShell `Select-Object`	Process text-based data based on patterns and expressions
`sed`	PowerShell `Replace-Text`	Substitute text within files based on patterns
`cut`	PowerShell `Select-String`	Extract specific columns or fields from text files
`join`	PowerShell `Merge-Object`	Combine data from multiple files based on common fields
`sort`	PowerShell `Sort-Object`	Sort data by specific columns or criteria
`uniq`	PowerShell `Remove-Duplicates`	Remove duplicate lines from a file

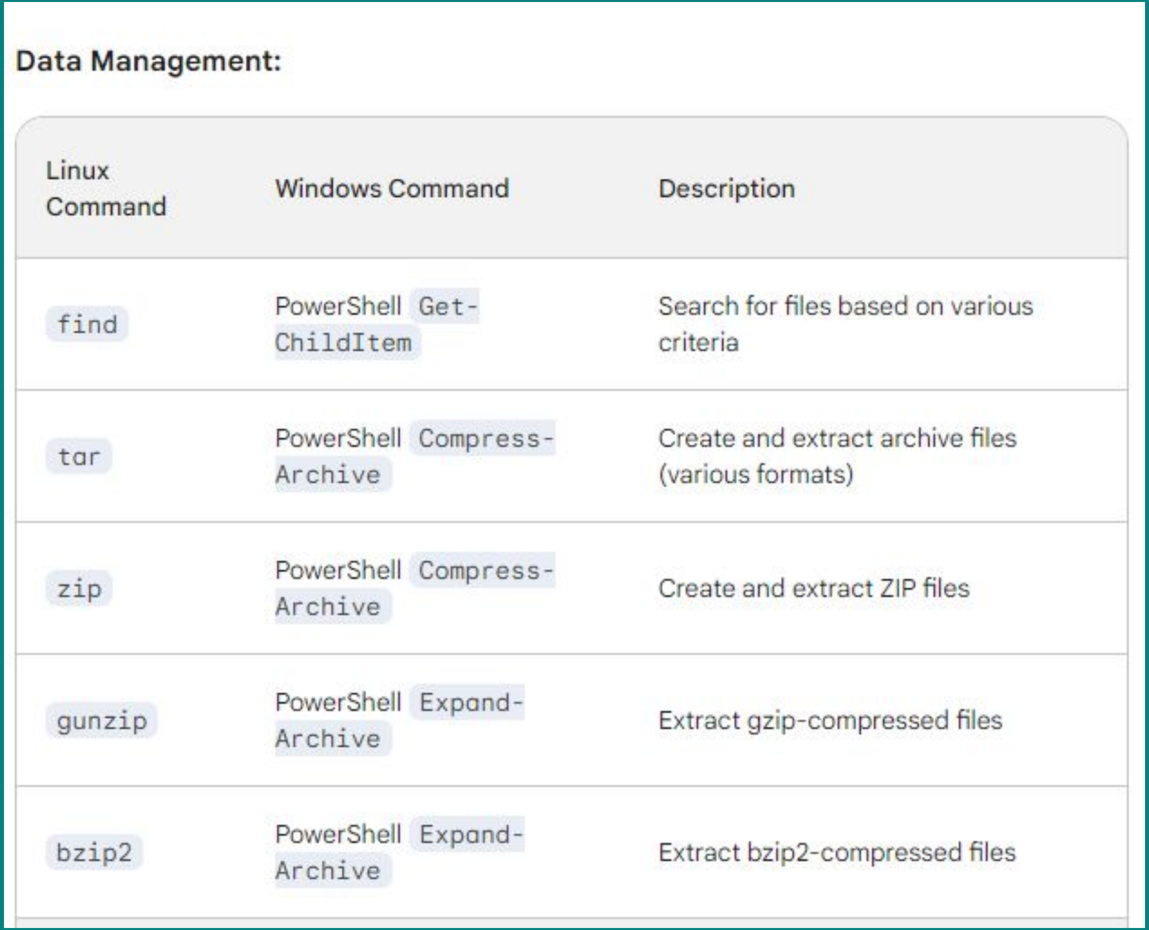
Important Linux Commands - Advance Part 2



Data Processing:

Linux Command	Windows Command	Description
`head`	PowerShell `Select-Top`	View the first few lines of a file
`tail`	PowerShell `Select-Last`	View the last few lines of a file
`wc`	PowerShell `Measure-Object`	Count lines, words, and characters in a file
`diff`	PowerShell `Compare-Object`	Compare the contents of two files
`checksum`	PowerShell `Get-FileHash`	Calculate checksums (e.g., MD5, SHA1) for file integrity

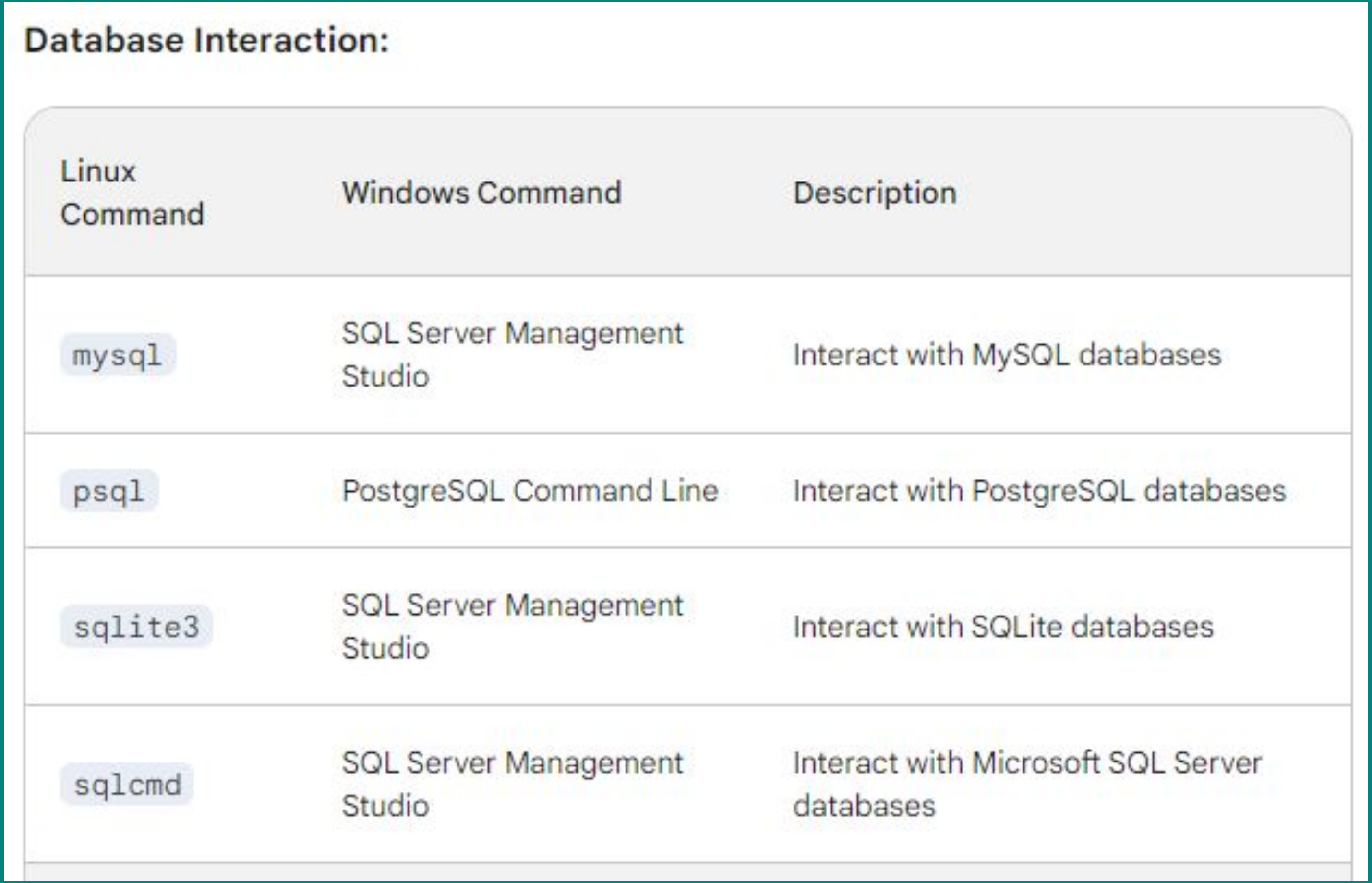
Important Linux Commands - Advance Part 3



Data Management:

Linux Command	Windows Command	Description
`find`	PowerShell `Get-ChildItem`	Search for files based on various criteria
`tar`	PowerShell `Compress-Archive`	Create and extract archive files (various formats)
`zip`	PowerShell `Compress-Archive`	Create and extract ZIP files
`gunzip`	PowerShell `Expand-Archive`	Extract gzip-compressed files
`bzip2`	PowerShell `Expand-Archive`	Extract bzip2-compressed files

Important Linux Commands - Advance Part 4



Database Interaction:

Linux Command	Windows Command	Description
`mysql`	SQL Server Management Studio	Interact with MySQL databases
`psql`	PostgreSQL Command Line	Interact with PostgreSQL databases
`sqlite3`	SQL Server Management Studio	Interact with SQLite databases
`sqlcmd`	SQL Server Management Studio	Interact with Microsoft SQL Server databases

Shell Scripting Programming - Part 1

1. If else statement

```
if <condition>; then
   # Statements to execute if the condition is true
else
   # Statements to execute if the condition is false
Fi
```

Example:

```
if [ $age -gt 18 ]; then
   echo "You are an adult."
else
   echo "You are not an adult."
Fi
```

```
if [[ $day =~ "${weekdays[@]}" ]]; then
  echo "It's a weekday! Time to work!"
elif [[ $day =~ "${weekends[@]}" ]]; then
  echo "It's the weekend! Enjoy your free time!"
else
  echo "Invalid day format!"
fi
```

Shell Scripting Programming - Part 2

Loops:

for loop:

```
for variable in values; do
   # Statements to execute for each value
done
```

Example:

```
for fruit in apple banana orange; do

   echo "I like $fruit."
done
```

while loop:

```
while <condition>; do
   # Statements to execute while the
condition is true
done
```

Example:

```
count=0
while [ $count -lt 5 ]; do
   echo "Count: $count"
   count=$((count+1))

  done
```

Shell Scripting Programming - Part 3

Functions:

```
function function_name () {
   # Function body
}
```

Example:

```
function greet () {
   name="$1"
   echo "Hello, $name!"
}

  greet John
```

Key Notes:

- `<condition>` can be any valid expression that evaluates to true or false.
- `<values>` can be a list of values, filenames, or variables.
- `$variable` within the loop iterates over each value.
- `$1`, `$2`, etc., represent arguments passed to a function.
- Use spaces around operators and semicolons (;).

Scheduling of Jobs in Linux

1. **Cron:**

- Cron is a built-in scheduling utility designed to execute commands or scripts at specific times or intervals.
- It uses a configuration file called `crontab` to define the schedule and commands to run.
- **Syntax:** The `crontab` file uses five fields separated by spaces:
  - Minute (0-59)
  - Hour (0-23)
  - Day of the month (1-31)
  - Month (1-12)
  - Day of the week (0-7, where 0 and 7 are Sunday)
  - Command to execute
- **Examples:**
  - Run a script every hour: `0 * * * * /path/to/script.sh`
  - Run a script at 2:30 AM daily: `30 2 * * * /path/to/script.sh`
  - Run a script every Monday at 8:00 PM: `0 20 * * 1 /path/to/script.sh`
- **Resources:**
  - Crontab man page: `man crontab`
  - How to schedule jobs using cron: https://www.redhat.com/sysadmin/linux-cron-command

Scheduling of Jobs in Linux

2. Systemd Timers:

- Systemd timers are another method for scheduling tasks in Linux, particularly on newer systems.
- They offer more advanced features like dependencies, unit files, and more flexible scheduling options.
- **Setup:** Create a unit file with the desired schedule and command, then enable and start the timer.
- **Resources:**
    - Systemd timers documentation: https://man7.org/linux/man-pages/man5/org.freedesktop.systemd1.5.html
    - Creating and managing systemd timers: https://wiki.archlinux.org/title/List_of_applications/Other

Choosing the Right Method:

- For simple scheduling tasks, cron is usually sufficient and easier to configure.
- For more complex scheduling needs with advanced features, systemd timers might be a better choice.

Additional Tips:

- Be cautious when using root privileges in scheduled jobs. Consider using less privileged accounts when possible.
- Log the output of your scheduled jobs for troubleshooting and monitoring purposes.
- Use descriptive names for your cronjobs or timer units for easier identification.

Remember to consult the specific documentation for your Linux distribution and chosen method for detailed instructions and troubleshooting steps.